Incrementally Constructing and Updating Constrained Delaunay Tetrahedralizations with Finite Precision Coordinates

Hang Si[1] and Jonathan Richard Shewchuk[2]

[1] Weierstrass Institute for Applied Analysis and Stochastics (WIAS), Mohrenstraße 39, 10117, Berlin, Germany
[2] Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, CA, 94720-1776, U.S.A.

Summary. Constrained Delaunay tetrahedralizations (CDTs) are valuable for generating meshes of nonconvex domains and domains with internal boundaries, but they are difficult to maintain robustly when finite-precision coordinates yield vertices on a line that are not perfectly collinear and polygonal facets that are not perfectly flat. We experimentally compare two recent algorithms for inserting a polygonal facet into a CDT: a bistellar flip algorithm of Shewchuk (*Proc. 19th Annual Symposium on Computational Geometry*, June 2003) and a cavity retriangulation algorithm of Si and Gärtner (*Proc. Fourteenth International Meshing Roundtable*, September 2005). We modify these algorithms to succeed in practice for polygons whose vertices deviate from exact coplanarity.

1 Introduction

A constrained Delaunay triangulation (CDT) is a variation of a Delaunay triangulation that is constrained to respect the boundary of a domain. CDTs in the plane were introduced by Lee and Lin [10]. Shewchuk [15, 20] generalized them to three or more dimensions. CDTs have optimality properties similar to those of Delaunay triangulations [10, 20]. Their ability to conform to domain boundaries makes them valuable for applications such as finite element simulation, computer graphics, and geographic information systems. Several algorithms for constructing CDTs have been proposed [17, 18, 19, 21, 22]. Some of them operate by constructing an ordinary Delaunay triangulation and then inserting polygonal boundaries one by one.

In this paper, we study how to incrementally update a three-dimensional CDT by inserting a polygon. This operation suffices to incrementally construct a CDT from a Delaunay triangulation of the vertices. We have implemented two such polygon insertion algorithms in the program `TetGen`: Shewchuk's flip-based algorithm [19] and Si and Gärtner's cavity retriangulation algo-

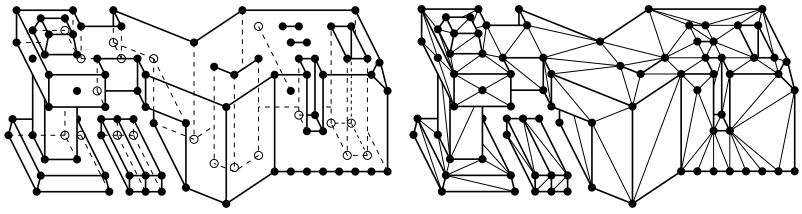

Fig. 1. A piecewise linear complex and its constrained Delaunay triangulation.

rithm [22]. We propose solutions to several issues that have been ignored in more theoretical papers, particularly the fact that vertices inserted on a common line are not always perfectly collinear, and the vertices of a polygonal facet are not always perfectly coplanar. We also suggest fixes to problems that arise when roundoff error causes the flip algorithm to perform flips in the wrong order. We experimentally compare the speed and robustness of the two polygon insertion algorithms.

2 Constrained Delaunay Triangulations

We assume that the reader is familiar with simplicial complexes and Delaunay triangulations of point sets in two and three dimensions. We model domains as *piecewise linear complexes* (PLCs), introduced by Miller et al. [11], which generalize polyhedra to permit interior boundaries and other constraints. In three dimensions, a PLC $\mathcal{X}$ is a set of vertices, edges, polygons, and polyhedra (not necessarily convex), collectively called *cells*, that satisfies the following properties. (1) The boundary of each cell in $\mathcal{X}$ is a union of cells in $\mathcal{X}$. (2) If two distinct cells $f, g \in \mathcal{X}$ intersect, their intersection is a union of cells in $\mathcal{X}$, all having lower dimension than at least one of f or g.

These conditions are intentionally permissive; for instance, they permit polygons, edges, and vertices to float in a polyhedral domain, or edges and vertices to float in a polygon, as Figure 1 shows. One purpose of these floating cells is to constrain how the PLC can be triangulated, so that boundary conditions may be accurately applied at those cells. The purpose of the polyhedra in $\mathcal{X}$ is to indicate which portions of space are interior to the domain and should be filled with tetrahedra.

The *underlying space* of a PLC $\mathcal{X}$, denoted $|\mathcal{X}|$, is $\bigcup_{f \in \mathcal{X}} f$, which is usually the domain to be triangulated. A *triangulation of* $\mathcal{X}$, also known in three dimensions as a *tetrahedralization of* $\mathcal{X}$, is a simplicial complex $\mathcal{T}$ such that (1) $\mathcal{X}$ and $\mathcal{T}$ have the same vertices, (2) every cell in $\mathcal{X}$ is a union of simplices in $\mathcal{T}$, and (3) $|\mathcal{T}| = |\mathcal{X}|$. An example appears in Figure 1.

A *mesh of* $\mathcal{X}$, also known as a *Steiner triangulation of* $\mathcal{X}$ or a *conforming triangulation of* $\mathcal{X}$, is a triangulation of $\mathcal{X} \cup S$, where $S \subset |\mathcal{X}|$ is a finite set of

Steiner points disjoint from the vertices in $\mathcal{X}$. In our parlance, a triangulation of $\mathcal{X}$ does not permit added vertices, whereas a mesh of $\mathcal{X}$ does.

A mesh $\mathcal{T}$ of $\mathcal{X}$ subdivides each polygon in $\mathcal{X}$ into triangles in $\mathcal{T}$, and each edge in $\mathcal{X}$ into edges in $\mathcal{T}$. We call the edges in a PLC *segments* to distinguish them from the edges in a mesh of the PLC. An edge in $\mathcal{T}$ included in a segment in $\mathcal{X}$ is called a *subsegment*. A triangle in $\mathcal{T}$ included in a polygon in $\mathcal{X}$ is called a *subpolygon*.

There are several definitions of constrained Delaunay triangulation [20]. For brevity, we omit the primary definition and give an equivalent, simpler definition. The *circumsphere* of a tetrahedron t is the unique sphere that passes through all four vertices of t. A *circumsphere* of a triangle s is any sphere that passes through all three vertices of s; there are infinitely many such circumspheres. A triangle s in a tetrahedralization $\mathcal{T}$ is said to be *locally Delaunay* if it is a face of fewer than two tetrahedra in $\mathcal{T}$, or it is a face of exactly two tetrahedra t_1 and t_2 and it has a circumsphere that encloses no vertex of t_1 nor t_2. Equivalently, the circumsphere of t_1 encloses no vertex of t_2. Equivalently, the circumsphere of t_2 encloses no vertex of t_1.

A triangulation $\mathcal{T}$ of a PLC $\mathcal{X}$ is a *constrained Delaunay triangulation* (CDT) of $\mathcal{X}$ if every triangle in $\mathcal{T}$ not included in a polygon in $\mathcal{X}$ is locally Delaunay. A crucial difference between an ordinary Delaunay triangulation and a CDT is that triangles included in PLC polygons are not required to be locally Delaunay, which frees the CDT to respect the PLC's polygons.

A *Steiner CDT* of $\mathcal{X}$ is a CDT of $\mathcal{X} \cup S$, where $S \subset |\mathcal{X}|$ is a set of Steiner points. Our meshing algorithms construct Steiner CDTs of input PLCs.

Every PLC in the plane has a CDT, which has no extra vertices not in the PLC. This fact gives CDTs an advantage over purely Delaunay meshes, which may require many extra vertices to force them to respect their domains. In three dimensions, the comparison between CDTs and Delaunay triangulations is more complicated. A difficulty of working with CDTs is that not every PLC has one; there even exist simple polyhedra that have no tetrahedralization at all. However, every PLC has Steiner CDTs; we can usually create one by adding a modest number of Steiner points; and the number required is typically far less than a purely Delaunay mesh of the PLC would require. In particular, for domains in which polygons meet at small dihedral angles, purely Delaunay meshes often have far more vertices than desired.

Updating a CDT is in some ways like updating a Delaunay tetrahedralization, but there are catches. The first catch is that every modification is done in the context of an underlying PLC. It is not usually possible to determine how a CDT will change when a vertex or polygon is inserted or deleted without knowing the PLC that determines it. Every incremental operation changes the PLC and the CDT together. The second catch is that the modified PLC might not have a CDT (or even a tetrahedralization).

Our algorithms rely on the *CDT Theorem*, which provides a useful sufficient condition for a PLC (or a polyhedron) to have a CDT. An edge $e \in \mathcal{T}$ is *strongly Delaunay* if there exists a ball whose boundary passes through the

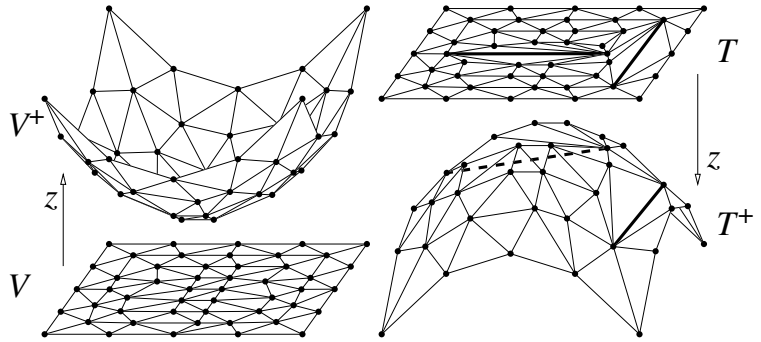

Fig. 2. Left: the parabolic lifting map. In this example, a two-dimensional vertex set V is lifted to a paraboloid in $\mathbb{R}^3$. The underside of the convex hull of the lifted vertices projects down to a Delaunay triangulation of V. Right: a lifted CDT, with the paraboloid inverted to more clearly show its topography. The bold edges are constraining edges that are not locally Delaunay. They are mapped to reflex edges of the lifted surface.

two vertices of e, but no other vertex of $\mathcal{T}$ lies in that ball. (This is a slightly stronger condition than e being Delaunay, which requires only that no vertex lie in the ball's interior.) A PLC is *edge-protected* if all its segments are strongly Delaunay. The CDT Theorem states that every edge-protected PLC has a CDT [20].

If a PLC is not edge-protected, it can be made edge-protected with the addition of carefully chosen Steiner points that subdivide its segments, yielding an augmented PLC $\mathcal{Y}$ that has a CDT. Some meshing algorithms choose these Steiner points so that $\mathcal{Y}$ does not have unreasonably short edges [18, 22]. One way to construct the CDT of $\mathcal{Y}$ is to construct a Delaunay triangulation of $\mathcal{Y}$'s vertices, then insert $\mathcal{Y}$'s polygons one by one. The CDT Theorem guarantees that these polygon insertions will succeed.

Delaunay triangulations and convex hulls are connected through the well-known *parabolic lifting map* of Seidel [13]. Let V be a set of vertices in $\mathbb{R}^3$ for which a Delaunay triangulation is sought. The lifting map sends each vertex in V to a vertex on a paraboloid in four-dimensional space, as Figure 2 (left) illustrates. Specifically, each vertex $\mathbf{v} = (v_x, v_y, v_z) \in V$ maps to a point $\mathbf{v}^+ = (v_x, v_y, v_z, v_x^2 + v_y^2 + v_z^2) \in \mathbb{R}^4$. The vertex $\mathbf{v}^+$ is the *lifted companion* of $\mathbf{v}$. Let V^+ be the set of lifted companions of the vertices in V. The Delaunay triangulation of V has the same combinatorial structure as the underside of the convex hull of V^+. Each tetrahedron on the underside of the convex hull of V^+ projects down to a tetrahedron in the Delaunay triangulation of V. This connection is used routinely to transform any $(d+1)$-dimensional convex hull algorithm into a d-dimensional Delaunay triangulation algorithm.

For a simplex s in $\mathbb{R}^3$, its lifted companion s^+ is the simplex embedded in $\mathbb{R}^4$ whose vertices are the lifted companions of the vertices of s. Let $\mathcal{T}^+ = \{s^+ : s \in \mathcal{T}\}$ be a tetrahedralization $\mathcal{T}$ lifted to the parabolic lifting map, as

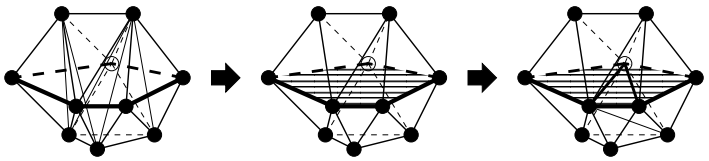

Fig. 3. Inserting a polygon (shaded) into a CDT. The simplices whose interiors intersect the polygon (left) are deleted; their union is a polyhedral cavity, which the polygon subdivides into two cavities (center). We retriangulate these cavities with their CDTs (right).

illustrated at right in Figure 2. If a triangular face s of two tetrahedra t_1 and t_2 in $\mathcal{T}$ is locally Delaunay, then the lifted triangle s^+ is *locally convex* in the sense that t_1^+ and t_2^+ meet at a dihedral angle, measured from above, of at most 180°.

The flip-based polygon insertion algorithm discussed in Section 3 requires the idea of a weighted CDT, which is similar to a CDT, but each vertex $\mathbf{v} \in \mathcal{X}$ is assigned a real-valued weight $w_{\mathbf{v}}$. A vertex $\mathbf{v}$ is lifted to a point $\mathbf{v}^+ = (v_x, v_y, v_z, v_x^2 + v_y^2 + v_z^2 - w_{\mathbf{v}})$; thus the weight $w_{\mathbf{v}}$ signifies the distance of $\mathbf{v}^+$ below the paraboloid. A triangulation $\mathcal{T}$ of a PLC $\mathcal{X}$ is a *weighted CDT* of $\mathcal{X}$ if every triangle $s \in \mathcal{T}$ not included in a polygon in $\mathcal{X}$ lifts to a locally convex triangle s^+. A triangle included in a PLC polygon is exempt from the requirement that it be locally convex; see the reflex ridges in $\mathcal{T}^+$ in Figure 2. A weighted CDT is identical to an ordinary CDT if all the weights are zero, but it can differ if some weights are nonzero. The flip-based polygon insertion algorithm linearly varies the weights of a PLC's vertices while using bistellar flips to maintain a weighted CDT of the PLC; see Section 3.2.

3 Polygon Insertion

To "insert a polygon into a CDT" is to take as input the CDT $\mathcal{T}$ of some PLC $\mathcal{X}$ and a new polygon f to insert, and produce the CDT $\mathcal{T}^f$ of $\mathcal{X}^f = \mathcal{X} \cup \{f\}$. It is only meaningful if $\mathcal{X}^f$ is a valid PLC—which implies that f's boundary is a union of segments in $\mathcal{X}$, among other things. It is only possible if $\mathcal{X}^f$ has a CDT.

A key observation is that when a polygon f is inserted, every simplex in $\mathcal{T}$ that respects f remains in $\mathcal{T}^f$. Let R be the union of tetrahedra in $\mathcal{T}$ whose interiors intersect f, as illustrated at left in Figure 3. Typically, R is a polyhedron which might not be convex; it might even have handles. A polygon insertion operation retriangulates the region R to respect f.

The new polygon f subdivides R into two cavities C_1 and C_2. In principle, we could construct $\mathcal{T}^f$ by computing CDTs of C_1 and C_2 and using them to replace the tetrahedra in R. Neither of the two algorithms we implemented are that straightforward, but they strive to achieve the same outcome.

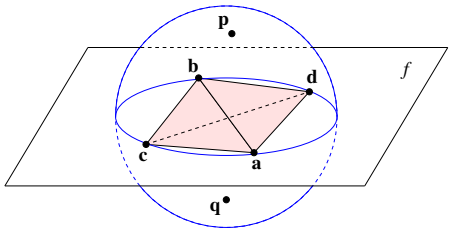

Fig. 4. A problem caused by a polygon f that is not perfectly flat: neither **ab** nor **cd** is a legal diagonal. The blue sphere is the circumsphere of **abcd**.

There is a notable difference between polygon insertion in theory and in practice, whose resolution is one of the main goals of this paper. Polygon insertion algorithms assume that polygons are flat—that is, all the vertices of a polygon are coplanar. However, in most implementations the vertex coordinates are floating-point numbers of fixed precision, and often a polygon's vertices are only approximately coplanar. This can cause a vexing problem illustrated in Figure 4. In this example, f is a polygon containing the vertices **a**, **b**, **c**, and **d**, which are not exactly coplanar but are nearly so. The vertices **p** and **q** lie above and below f, respectively. Moreover, **p** and **q** both lie inside the circumsphere of the tetrahedron **abcd**. The edge **ab** cannot be in a CDT because of the vertices **c**, **d**, and **p**; for example, if we construct tetrahedra **abcp** and **abdp**, the face **abp** will not be locally Delaunay. Symmetrically, the edge **cd** cannot be in a CDT because of the vertices **a**, **b**, and **q**; if we construct **acdq** and **bcdq**, **cdq** will not be locally Delaunay. There is no way to triangulate the shaded quadrilateral region of f that is compatible with completing a three-dimensional CDT on both sides of f.

In this section, we describe two polygon insertion algorithms and compare their performance in practice. Neither algorithm explicitly computes a two-dimensional triangulation of f in advance; in both algorithms, a CDT of f emerges naturally as a byproduct of the three-dimensional triangulation.

3.1 Polygon Insertion by Cavity Retriangulation

The faces bounding the cavities C_1 and C_2 are triangles, except f. The cavity retriangulation algorithm [22] retriangulates C_1 and C_2 separately.

In principle, we could construct a CDT of each cavity by gift-wrapping [17]. However, the incremental vertex insertion algorithm is faster in practice. Unfortunately, the latter algorithm constructs Delaunay triangulations, not CDTs. In practice, the CDT and the Delaunay triangulation are identical sufficiently often that it pays to use the incremental algorithm and fix the violated constraints when they occur.

The cavity retriangulation algorithm of Si and Gärtner [22] first constructs a Delaunay triangulation (unconstrained) $\mathcal{D}$ of the vertices of a cavity C. The algorithm identifies which triangular faces of C are included in $\mathcal{D}$. Because

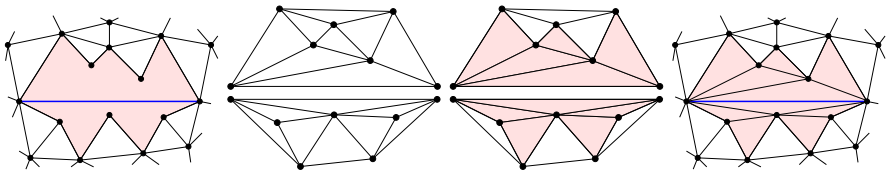

Fig. 5. The cavity retriangulation algorithm, illustrated in two dimensions. From left to right, the two initial cavities C_1 and C_2 separated by a segment; the initial Delaunay triangulations $\mathcal{D}_1$ and $\mathcal{D}_2$; triangles of $\mathcal{D}_1$ and $\mathcal{D}_2$ are classified as "inside" or "outside"; and the new triangulations of C_1 and C_2.

f lies on the boundary of the convex hull of C, and because every edge of f is strongly Delaunay, we can also identify a subset of triangles in $\mathcal{D}$ that constitute a triangulation of f.

In practice, the cavity triangulation $\mathcal{D}$ often contains every triangular face of the cavity C. Frequently C is not convex; by removing from $\mathcal{D}$ the simplices that lie outside C, we reveal a triangulation $\{t \in \mathcal{D} : t \subseteq C\}$ that fills the cavity, as illustrated in Figure 5. We graft this triangulation onto the main mesh $\mathcal{T}$.

Occasionally, the Delaunay triangulation $\mathcal{D}$ of C's vertices fails to include a CDT of C. The first solution one would think of is to repair $\mathcal{D}$ to make it a CDT, but we forgo that for a solution that is easier to implement robustly. Our strategy is to enlarge the cavity C until all its boundary triangles are included in a Delaunay triangulation of its vertices, and use subsequent calls to the polygon insertion algorithm to repair the deviations of the Delaunay triangulation from the CDT.

Let s be a triangular face of C that is absent from the Delaunay triangulation $\mathcal{D}$. Let t be the tetrahedron of the complete mesh $\mathcal{T}$ that has s for a face and lies outside the cavity C. We enlarge the cavity by setting $C \to C \cup t$, and update the list of C's boundary faces, which no longer contains s. If t has a vertex not already in the triangulation $\mathcal{D}$, we update $\mathcal{D}$ by inserting that vertex. Again, we test whether $\mathcal{D}$ contains every triangular face of C; if not, we repeat the process of enlarging C at a missing face until every face is included. This loop must terminate eventually because $\mathcal{T}$ has only a finite number of vertices, and the loop succeeds when C is the convex hull of C's vertices. We graft $\{t \in \mathcal{D} : t \subseteq C\}$ into $\mathcal{T}$.

The Delaunay triangulation $\mathcal{D}$ does not necessarily respect the polygons that pass through its interior, so this cavity retriangulation algorithm sometimes removes polygons that had previously been inserted. These polygons are added again to the queue, and are reinserted immediately after f is inserted.

If f's vertices are not exactly coplanar, practical problems arise that do not exist in the idealized algorithm where f is perfectly flat. $\mathcal{D}$ can include extremely thin tetrahedra whose four vertices all lie on f, which we call *shim tetrahedra*. Moreover, some of f's vertices may lie slightly in the interior of C.

Shim tetrahedra cannot be permitted to survive in the final mesh. We delete from $\mathcal{D}$ all the shim tetrahedra necessary to expose all f's vertices on the boundary of $\mathcal{D}$. We can often remove all the other shim tetrahedra too, but we must keep shim tetrahedra that appear in both cavity triangulations $\mathcal{D}_1$ and $\mathcal{D}_2$ for the two cavities C_1 and C_2.

There are two problems that can arise because the cavity triangulations $\mathcal{D}_1$ and $\mathcal{D}_2$ must be compatible where they meet each other. First, in Figure 4, the two cavity triangulations are incompatible because they triangulate f differently. The cavity triangulation on the bottom side of f includes the triangles **abc** and **abd**, whereas the cavity triangulation on the top side includes the triangles **acd** and **bcd**; therefore, the two cavity triangulations overlap each other, both occupying the space in the tetrahedron **abcd**. Second, if the circumsphere of **abcd** were empty (neither **p** nor **q** were present), then the shim tetrahedron **abcd** must be present to bridge the triangulations of the two cavities. (The ideal circumstance would be to have only one of **p** or **q** present. For instance, if **p** is present but no vertex lies in the bottom half of the circumsphere of **abcd**, then the two cavity triangulations agree on the shared boundary triangles **acd** and **bcd**, with no shim tetrahedron.)

We resolve both these problems by inserting a Steiner point on the polygon f according to the usual encroachment rules of Delaunay refinement algorithms [16], treating one of the triangular faces of **abcd** as an encroached triangle. Typically we insert a new vertex at the circumcenter of a face of **abcd**, but if that circumcenter encroaches upon an edge of f, the edge may be split instead. (Note that the vertices of **abcd** are approximately cocircular, so it does not matter much which face we choose; all four have approximately the same circumcenter.)

3.2 Polygon Insertion by Bistellar Flips

The flip algorithm of Shewchuk [19] retriangulates R by performing a sequence of tetrahedral bistellar flips, specifically 2-to-3, 3-to-2, and 4-to-4 flips, that transform $\mathcal{T}$ into $\mathcal{T}^f$. We call a triangulation simplex a *crossing simplex* if it is included in R and has at least one vertex on each side of f. Every simplex deleted by a flip is a crossing simplex. When no crossing simplex survives, the updated triangulation is $\mathcal{T}^f$.

The algorithm chooses flips that remove crossing triangles, but the order in which it performs such flips is crucial: if the flips are performed in the wrong sequence, the algorithm can get stuck in a configuration from which no further progress is possible. To guide the flipping process, we maintain at all times the invariant that the tetrahedra in R form a weighted CDT of the vertices in R. The weights of the vertices change continuously and linearly as a function of a time variable τ, and we dynamically update the weighted CDT to reflect these weight changes. A priority queue stores flips keyed according to the times when they occur. The algorithm operates by repeatedly dequeueing the earliest flip and performing it if it is still possible.

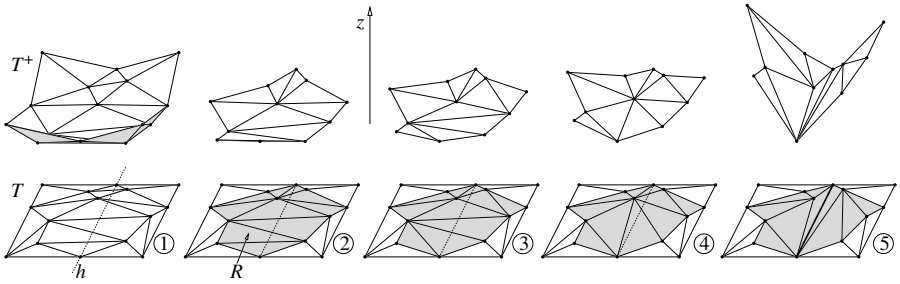

Fig. 6. A two-dimensional example: inserting a segment into a CDT.

Let h be the plane in $\mathbb{R}^3$ that includes f (and cuts R into two cavities). Call the vertices on one side of h *left vertices*, and the vertices on the other side *right vertices*. Vertices on h are neither. By definition, a crossing simplex has at least one left vertex and at least one right vertex.

The flip algorithm decreases the weights of the left and right vertices at a rate linearly proportional to the time τ and their distances from h. The weights of the vertices on h remain fixed at zero, and simplices outside R are ignored entirely. The vertices of f have weight zero, even if some of them do not lie exactly on h. On the parabolic lifting map, the effect is that the heights of the left and right vertices in $\mathbb{R}^4$ increase linearly, as illustrated in Figure 6. The lifted left vertices in $\mathbb{R}^4$ undergo an affine transformation, so no flips occur among simplices solely on the left side of h; symmetrically, the simplices on the right side of h are stable. But crossing simplices cannot survive in the weighted CDT in the limit as the heights of the lifted vertices go to infinity, while the weights of f's vertices remain fixed at zero.

Whenever two crossing tetrahedra reach a state where their lifted companions in $\mathbb{R}^4$ lie on a common non-vertical hyperplane, the algorithm performs a flip that replaces them, and perhaps some adjoining tetrahedra, with new tetrahedra. When the time τ reaches infinity, no crossing simplex survives, so f is represented as a union of mesh triangles and the algorithm is complete.

Figure 7 gives pseudocode for the flip algorithm. Lines 11 and 12 raise several numerical issues. It suffices to change the weights of the left vertices only, while fixing the weights of the right vertices at zero—this modification yields an algorithm that is equivalent in theory, and suffers less from roundoff error in practice. Let $\mathbf{v} = (v_x, v_y, v_z)$ be a left vertex in $\mathcal{T}$ that lies in R. In $\mathbb{R}^4$, the lifted companion of $\mathbf{v}$ is

$$\mathbf{v}^+ = (v_x, v_y, v_z, v_x^2 + v_y^2 + v_z^2 - w_{\mathbf{v}}),$$

whose weight and height change linearly with time according to the identity

$$w_{\mathbf{v}} = -\tau \cdot d(\mathbf{v}, h),$$

where $d(\mathbf{v}, h)$ is a measure of the distance from $\mathbf{v}$ to h. We recommend not using the Euclidean distance, whose computation involves a square root and the

```
FlipInsertPolygon(𝒳, 𝒯, f)
{ 𝒳 is a PLC. 𝒯 is its CDT. f is a polygon to insert. }
1       Q ← an empty priority queue
2       for each crossing triangle s in 𝒯
3           Certify(s)
4       while the priority queue Q is not empty
5           Remove ⟨s′, τ⟩ with minimum τ from Q
6           if s′ is still a triangle in 𝒯
7               Flip(𝒯, s′)
8               for each crossing triangle s on the boundary of the flipped volume
9                   Certify(s)

Certify(s)
10      Let t1 and t2 be the tetrahedra that share the face s
11      If the interior of t1+ will be below the affine hull of t2+ at time ∞
12          τ ← the time at which t1+ and t2+ are cohyperplanar
13          Insert ⟨s, τ⟩ into priority queue Q
```

Fig. 7. Algorithm for inserting a polygon into a CDT. Works if $\mathcal{T}$ is the CDT of the initial PLC $\mathcal{X}$, and the final PLC $\mathcal{X}^f = \mathcal{X} \cup \{f\}$ is a valid complex. Line 12 uses Equation (1), and Line 11 checks whether the denominator of that expression is positive. The subroutine Flip in Line 7 performs a 2-3, 3-2, or 4-4 flip that removes the triangle s' from $\mathcal{T}$; see Shewchuk [19] for pseudocode.

consequent roundoff error; instead, our implementation calculates the weight

$$w_{\mathbf{v}} = -\tau \cdot \textsc{Orient3D}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{v}), \text{ where}$$

$$\textsc{Orient3D}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{v}) = \det \begin{bmatrix} \mathbf{a}_x - \mathbf{v}_x & \mathbf{a}_y - \mathbf{v}_y & \mathbf{a}_z - \mathbf{v}_z \\ \mathbf{b}_x - \mathbf{v}_x & \mathbf{b}_y - \mathbf{v}_y & \mathbf{b}_z - \mathbf{v}_z \\ \mathbf{c}_x - \mathbf{v}_x & \mathbf{c}_y - \mathbf{v}_y & \mathbf{c}_z - \mathbf{v}_z \end{bmatrix}$$

and $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are three non-collinear vertices of f, oriented so that Orient3D returns a positive value if $\mathbf{v}$ is a left vertex. The quantity $\textsc{Orient3D}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{v})$ is six times the signed volume of the tetrahedron $\mathbf{abcv}$, and it is proportional to the Euclidean distance from $\mathbf{v}$ to h but can be calculated more accurately. In our implementation, it benefits from the accuracy of our orientation predicate [14]. The choice of the vertices $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ can make a difference, especially if f is not perfectly flat; we recommend choosing them to approximately maximize the area of $\mathbf{abc}$.

To compute the time τ when a flip takes place, let $t_1 = \mathbf{xyzv}$ and $t_2 = \mathbf{xyzw}$ be two tetrahedra that share a crossing triangle $s = \mathbf{xyz}$ as a common face. A flip might take place when the lifted companions of t_1 and t_2 lie on a common hyperplane; thus

$$\begin{aligned} 0 &= \textsc{Orient4D}(\mathbf{x}^+, \mathbf{y}^+, \mathbf{z}^+, \mathbf{v}^+, \mathbf{w}^+) \\ &= \textsc{InSphere}(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{v}, \mathbf{w}) + \tau \cdot \textsc{Orient4D}(\mathbf{x}^\ddagger, \mathbf{y}^\ddagger, \mathbf{z}^\ddagger, \mathbf{v}^\ddagger, \mathbf{w}^\ddagger) \end{aligned}$$

where ORIENT4D is a 4×4 determinant defined by analogy to ORIENT3D, $\mathbf{v}^\ddagger = (v_x, v_y, v_z, \text{ORIENT3D}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{v}))$ if $\mathbf{v}$ is a left vertex and $\mathbf{v}^\ddagger = (v_x, v_y, v_z, 0)$ otherwise, and

$$\text{INSPHERE}(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{v}, \mathbf{w}) = \det \begin{bmatrix} \mathbf{x}_x - \mathbf{w}_x & \mathbf{x}_y - \mathbf{w}_y & \mathbf{x}_z - \mathbf{w}_z & (\mathbf{x}_x - \mathbf{w}_x)^2 + (\mathbf{x}_y - \mathbf{w}_y)^2 + (\mathbf{x}_z - \mathbf{w}_z)^2 \\ \mathbf{y}_x - \mathbf{w}_x & \mathbf{y}_y - \mathbf{w}_y & \mathbf{y}_z - \mathbf{w}_z & (\mathbf{y}_x - \mathbf{w}_x)^2 + (\mathbf{y}_y - \mathbf{w}_y)^2 + (\mathbf{y}_z - \mathbf{w}_z)^2 \\ \mathbf{z}_x - \mathbf{w}_x & \mathbf{z}_y - \mathbf{w}_y & \mathbf{z}_z - \mathbf{w}_z & (\mathbf{z}_x - \mathbf{w}_x)^2 + (\mathbf{z}_y - \mathbf{w}_y)^2 + (\mathbf{z}_z - \mathbf{w}_z)^2 \\ \mathbf{v}_x - \mathbf{w}_x & \mathbf{v}_y - \mathbf{w}_y & \mathbf{v}_z - \mathbf{w}_z & (\mathbf{v}_x - \mathbf{w}_x)^2 + (\mathbf{v}_y - \mathbf{w}_y)^2 + (\mathbf{v}_z - \mathbf{w}_z)^2 \end{bmatrix}.$$

Hence,

$$\tau = -\frac{\text{INSPHERE}(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{v}, \mathbf{w})}{\text{ORIENT4D}(\mathbf{x}^\ddagger, \mathbf{y}^\ddagger, \mathbf{z}^\ddagger, \mathbf{v}^\ddagger, \mathbf{w}^\ddagger)}. \tag{1}$$

If the denominator $\text{ORIENT4D}(\mathbf{x}^\ddagger, \mathbf{y}^\ddagger, \mathbf{z}^\ddagger, \mathbf{v}^\ddagger, \mathbf{w}^\ddagger)$ is negative, then the triangular face s^+ will remain locally convex until (and including) time $\tau = \infty$, so Line 11 of FLIPINSERTPOLYGON declines to enqueue the potential flip. It is also possible that the denominator is zero, or so tiny that the quotient overflows (yielding an infinity in IEEE floating point), in which case we can also discard the potential flip, because some other flip will delete one of the tetrahedra earlier.

When no crossing simplex survives and f is recovered, we implicitly restore the weights of the left vertices to zero. If f is not perfectly flat, the two problems described at the end of Section 3.1 can manifest for the flip algorithm too. The flip algorithm can produce shim tetrahedra along f. More interesting is the circumstance depicted in Figure 4. The flip algorithm increases the weight of **q** to infinity while leaving the weights of the other five vertices at zero, and it produces the tetrahedra **acdp**, **bcdp**, **acdq**, and **bcdq**. But when we restore the weight of **q** to zero, we discover that the triangle **cdq** is not locally Delaunay, and so $\mathcal{T}^f$ is not a CDT.

Both problems (surviving shim tetrahedra and triangles that are not locally Delaunay) can sometimes be repaired by a 3-2 flip that removes one of the edges **cd** or **ab**, or a 4-4 flip that removes one and creates the other. If not, we resolve them by inserting Steiner points as described at the end of Section 3.1. If a shim tetrahedron **abcd** genuinely belongs in the CDT of $\mathcal{X}^f$, a flip is no solution; we eliminate the tetrahedron by inserting a new vertex on f. In the circumstance depicted in Figure 4, as we discuss in Section 3.1, there exists no triangulation of $\mathcal{X}^f$ whose unconstrained triangular faces are all locally Delaunay, so we insert a new vertex on f.

3.3 Implementation Notes

We have implemented both algorithms in version 1.5.0 of the tetrahedral Delaunay mesh generator `TetGen` (http://www.tetgen.org). We use a tetrahedron-based data structure [1] to represent the tetrahedralization, and a triangle-edge data structure [12] to represent cavity boundaries as surface

meshes. Each data structure has pointers into the other. For quick searching, every vertex has a pointer to one of the tetrahedra adjoining it. The tetrahedron data structure has flags that indicate which edges and triangles are subsegments or subpolygons, so we can quickly determine when a subsegment or subpolygon is deleted.

A program switch selects one of the two polygon insertion algorithms. The insertion of a polygon f begins with finding triangles in $\mathcal{T}$ whose three vertices all lie on f. Each of these triangles is either inside or outside f; they cannot cross f's boundary because the edges of f are strongly Delaunay. The algorithm then determines whether these triangles already form a triangulation of f, by a depth-first search from f's edges. If they do, then $\mathcal{T}_f = \mathcal{T}$ and the algorithm is done. Otherwise, we compute one or more *missing regions* [21], subsets of f with connected interiors. `TetGen` inserts one missing region at a time, not one polygon at a time.

We use our correct implementations of the geometric predicates Orient3D, InSphere, and Orient4D [14] in both the flip and cavity retriangulation algorithms; these suffice to guarantee the numerical robustness of the latter algorithm. To compute a cavity R, we implemented a robust triangle-edge intersection test [9] that relies on a combination of Orient3D tests. We construct all Delaunay triangulations (of the initial vertex set and all cavities) with the well-known incremental insertion algorithm [2, 23, 5] and the same geometric predicates. A simple symbolic perturbation scheme [7, 19] perturbs the vertex weights from zero to infinitesimal so that the InSphere test never returns a zero; thus there is always one canonical Delaunay tetrahedralization of any point set, and at most one canonical CDT of any PLC.

In the cavity retriangulation algorithm (Section 3.1), the incremental insertion algorithm uses the simplest walking algorithm for point location [6], because the cavities are usually small.

When the cavity retriangulation algorithm must insert a new vertex as described at the end of Section 3.1 to cope with a polygon that is not perfectly flat, it restores the original cavity triangulation before inserting the vertex. Our implementation does not delete the original tetrahedra until a new cavity triangulation is successfully computed without shim tetrahedra or triangles that are not locally Delaunay.

Our implementation of the flip algorithm required several adjustments of the algorithm to compensate for roundoff error in computing the flip times in Equation (1). Unfortunately, it would be difficult to make these computations completely robust; a correct ordering of flips would require us to exactly compare two quotients of the form (1) and to account properly for the infinitesimal vertex weight perturbations when two flips are simultaneous. This can be done with exact arithmetic, but we are not willing to incur the computational cost. Exact predicates can be made much faster with floating-point filters [8, 14], but we did not have time to derive the forward error bounds for these complicated polynomials.

One change we made was to ensure that the flip algorithm never removes a tetrahedron not in the cavity R. Because of floating-point roundoff error or our failure to account for the symbolic weight perturbations in our flip time computations, the algorithm occasionally tries to remove a crossing triangle with a 4-4 or 3-2 flip that also removes one or two tetrahedra that are not in R. In theory, this should never happen; all tetrahedra not in R should survive in the weighted CDT as the weights change. We modified the flip algorithm to discard any flip that would remove a tetrahedron not in R.

A second change copes with our discovery that, because of roundoff error or our unperturbed flip time computations, sometimes a pair of non-crossing tetrahedra appear in the cavity R that share a triangular face that is not locally Delaunay. In theory, this should never happen. We modified the flip algorithm so that whenever it produces a non-crossing tetrahedron, it tests the tetrahedron's non-crossing neighbors to ensure the shared face is locally Delaunay, and flips it if it is not.

A third, more complicated change compensates for flips that come off the priority queue in the wrong order. Because we compute the flip times with roundoff error, the flip with the least time sometimes cannot be performed until another flip with slightly greater time is performed. When we remove a flip from the priority queue that cannot be performed even though both tetrahedra are still in the mesh, we place the invalid flip in a list L. Every time we successfully perform a flip, we check every flip in L to see if it can now be performed, and perform it if possible; we also discard from L any flip whose two defining tetrahedra t_1 and t_2 do not both still exist. In practice, the list L is usually empty and never grows long.

We implemented the priority queue with the C++ standard library template `std::priority_queue<T, Container, Compare>`.

3.4 An Experimental Comparison

We compared the two polygon insertion algorithms in version 1.5.0 of `TetGen`, compiled by version 4.4 of `g++` with the `-O3` optimization level. Tests were performed on a laptop with a 2.8 GHz Intel CPU and 4 GB memory.

To measure how the two algorithms behave with respect to the number of deleted tetrahedra, we randomly generate n vertices distributed near, but not on, the x-y plane, and we specify a large rectangle in the x-y plane that cuts entirely through their convex hull, as illustrated in Figure 8. We construct the Delaunay tetrahedralization of the vertex set, including the four vertices of the rectangle, then insert the rectangle.

Figure 9 shows the running times for the two algorithms to insert a rectangle into differently sized cavities. At this scale, the running times of both algorithms appear to be linear in the number of crossing tetrahedra. For larger cavities, the cavity retriangulation algorithm is about twice as fast as the flip algorithm, but for very small cavities the flip algorithm is somewhat faster.

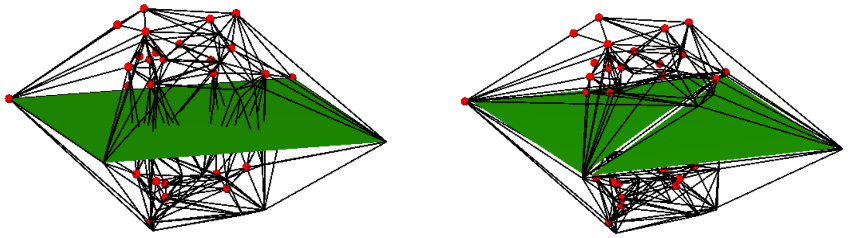

Fig. 8. A test example: inserting a rectangle (shaded) into the Delaunay triangulation of a random vertex set.

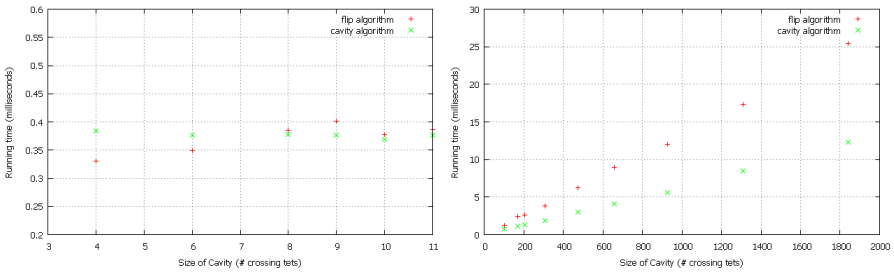

Fig. 9. Running times for the flip and cavity retriangulation algorithms to insert a rectangle into a Delaunay triangulation of a random point set. The horizontal axis shows the number of crossing tetrahedra deleted from the evacuated region R; in the right plot this number varies from 102 to 1,840. The vertical axis shows the running time in milliseconds.

Our second experiment measures the performance of the two algorithms in constructing CDTs. The input PLCs are freely available from the mesh repository provided by INRIA's GAMMA group (http://www-roc.inria.fr/gamma/gamma/gamma.php). Statistics are tabulated in Table 1. The two shaded rows **(14)** and **(15)** report the running times of polygon recovery by the cavity retriangulation and flip algorithms, respectively. The running times differ by less than 6%, and less than 1% of the total CDT construction time.

The cavity retriangulation algorithm is slightly faster when the mean cavity size, row **(11)**, is large, but the flip algorithm is slightly faster when the mean cavity size is small. To help explain this effect, Table 2 shows what proportion of each algorithm's running time was spent on several tasks. The cavity retriangulation algorithm spends most of its time constructing Delaunay triangulations of cavities, but the smaller the cavity, the greater the proportion of its time it spends on auxiliary tasks such as identifying cavity faces in the cavity triangulation $\mathcal{D}$ and deleting from $\mathcal{D}$ the tetrahedra that are not in the cavity.

Out of the 229 PLCs we tested, eight required that Steiner points be inserted because of polygons that are not perfectly flat during the polygon insertion stage (as distinct from the segment recovery stage, in which most

	DEM004	DEM005	anc101	monster4	mohne-a	thru-maz	cognit	nasty_ch
1	1,180	1,128	1,378	1,392	2,760	2,781	2,992	8,630
2	2,372	2,280	2,772	2,784	5,560	5,622	5,792	17,782
3	2,615	1,551	249	1,024	6,995	749	8,490	42,408
4	11,810	7,470	5,654	7,198	27,956	8,378	39,386	149,757
5	7,602	5,382	3,270	4,832	19,550	7,120	23,000	102,604
6	24	25	60	71	41	87	148	112
7	9	5	15	11	6	15	9	14
8	133	206	386	730	296	167	232	290
9	169	261	391	860	397	266	460	456
10	55	49	63	159	100	28	23	54
11	28	24	33	41	19	7	5	9
12	0.0685	0.0631	0.0795	0.0724	0.1483	0.1496	0.0948	1.4972
13	0.1332	0.0806	0.0288	0.1399	0.3273	0.0436	0.3254	3.1978
14	0.0255	0.0184	0.0224	0.0432	0.0429	0.0293	0.0472	0.2311
15	0.0254	0.0195	0.0237	0.0440	0.0427	0.0285	0.0456	0.2300
16	0.2449	0.1759	0.1433	0.2701	0.5667	0.2476	0.5385	5.3380

Table 1. Statistics for CDT construction. The columns list input PLCs. The rows report **(1)**-number of input vertices; **(2)**-number of input polygons; **(3)**-number of Steiner points added; **(4)**-number of output tetrahedra; **(5)**-number of output subpolygons (triangles covering input polygons); **(6)**-total number of recovered subpolygons; **(7)**-maximum number of recovered subpolygons in a single missing region; **(8)**-total number of flips performed by the flip algorithm; **(9)**-total number of crossing tetrahedra; **(10)**-maximum number of crossing tetrahedra for a single missing region; **(11)**-mean crossing tetrahedra per missing region; **(12)**-time to construct the initial Delaunay tetrahedralization (seconds); **(13)**-time for segment recovery; **(14)**-total time used by the cavity retriangulation algorithm; **(15)**-total time used by the flip algorithm; **(16)**-total time to construct the CDT, with the cavity retriangulation algorithm. All running times are in seconds.

PLCs require Steiner points to make them edge-protected). One PLC, called `90-Bend_cut` in the INRIA GAMMA collection, required the insertion of 103 vertices during polygon insertion: 27 on polygons, and 76 on segments due to encroachment. Our vertex insertion strategy was always successful at finding a valid Steiner CDT.

We observed one PLC for which the flip algorithm failed because it got stuck at a triangulation that was not a CDT, but none of the flips on the queue could be performed. This configuration arose after flips occurred in the wrong order because of rounding errors in the flip time computations or our failure to account for the weight perturbations in these computations. The PLC includes six vertices at the corners of a rectangular prism, with all six vertices lying on a common sphere. A seventh vertex inside that sphere, but outside the prism, anchors a set of tetrahedra that fan out to four of the prism's facets. Because of an out-of-order flip, the triangulation edges on the boundary of the prism assume a configuration isomorphic to the boundary of a well-known

	DEMO04	DEMO05	anc101	monster4	mohne-a	thru-maz	cognit	nasty_ch
1	74.7%	72.5%	79.0%	80.0%	72.1%	66.4%	60.5%	59.2%
2	8.5%	9.2%	6.5%	7.1%	8.4%	11.2%	11.2%	11.0%
3	0.7%	0.9%	0.5%	0.3%	1.0%	1.9%	2.5%	1.7%
4	3.4%	4.0%	2.4%	2.0%	5.0%	5.3%	7.3%	6.8%
5	65.8%	70.3%	64.4%	64.2%	58.6%	61.2%	55.5%	48.6%
6	25.5%	23.9%	29.1%	28.3%	33.4%	28.5%	31.3%	34.3%

Table 2. Proportion of the running time spent performing different tasks. For the cavity retriangulation algorithm, the rows report time spent **(1)**-constructing a Delaunay tetrahedralization of a cavity; **(2)**-identifying cavity faces in the tetrahedralization; **(3)**-identifying tetrahedralization faces on the polygon; **(4)**-removing tetrahedra from nonconvex cavities and merging the cavity triangulation with the mesh. For the flip algorithm, the rows report time spent **(5)**-computing flip times and placing potential flips on the priority queue; **(6)**-dequeueing flips from the priority queue and performing them.

polyhedron that has no tetrahedralization, called *Schönhardt's polyhedron*, whereupon the flip algorithm becomes stuck; none of the remaining flips on the priority queue can execute.

Because of this failure case, we recommend the cavity retriangulation algorithm over the flip algorithm unless resources are available to implement the latter algorithm's priority queue comparisons with exact arithmetic and symbolic weight perturbations.

4 Discussion

The two polygon insertion algorithms we have implemented—cavity retriangulation and flipping—are fast and comparable in speed. As we have implemented them, the cavity retriangulation algorithm has the advantage that it has been entirely reliable, whereas the flip algorithm can fail because we have not been willing to invest the effort to make its priority queue numerically robust. The cavity retriangulation algorithm requires only orientation and "insphere" tests, which are much easier to implement correctly for floating-point coordinates than the priority queue comparisons of the flip algorithm. The only strike against the cavity retriangulation algorithm is that we have no guarantee that it runs in polynomial time, though we have never seen it run slowly in practice.

Polygon vertices that are not perfectly coplanar introduce problems that are not merely problems of accuracy and are not easily finessed away. It is undesirable that we must sometimes insert new vertices on polygons when roundoff yields a domain that does have a CDT, but the condition that each unconstrained triangular face be locally Delaunay is a firm requirement that we do not believe could be relaxed without risking the loss of any useful structure as the mesh is refined. A promising direction of future research is to

fix some of these problems without vertex insertions by instead maintaining a weighted CDT and assigning small weights to vertices on polygons, in the manner of sliver exudation algorithms [3], to ensure that a weighted CDT exists despite deviations from flatness. We observe that the problems discussed in Section 3 occur only where four vertices on a polygon are nearly cocircular (albeit not perfectly coplanar). One could choose vertex weights so that no four vertices on a polygon are close to having coplanar lifted companions in $\mathbb{R}^4$ (the weighted equivalent of cocircularity).

We would also like to find an alternative to the flip algorithm that does not require a priority queue or geometric predicates more complicated than orientation and insphere tests, but has a good bound on its asymptotic running time. We suspect that fast algorithms for vertex deletion related to Chew's algorithm [4] might be a basis for a fast polygon insertion algorithm.

Another problem with refining constrained Delaunay meshes is that the addition of a vertex to a PLC can yield a new PLC that has no CDT. Sometimes it is necessary to insert multiple vertices to return the PLC to a state where it has a CDT. The question is how to update the CDT quickly. We believe that the cavity retriangulation algorithm can be modified to handle this circumstance.

There are several open questions. We do not know a theoretical worst-case running time for our cavity retriangulation algorithm. Nor do we know an algorithm that guarantees that a three-dimensional PLC has a CDT by inserting only a polynomial number of Steiner points on the PLC segments [18, 22]. These issues might not be very relevant in practice, as the algorithms are nearly always quite efficient.

References

1. Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. *Triangulations in CGAL.* Computational Geometry: Theory and Applications **22**(1–3):5–19, May 2002.
2. Adrian Bowyer. *Computing Dirichlet tessellations.* The Computer Journal **24**(2):162–166, 1981.
3. Siu-Wing Cheng, Tamal Krishna Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. *Sliver exudation.* Journal of the Association for Computing Machinery **47**(5):883–904, September 2000.
4. L. Paul Chew. *Building Voronoi diagrams for convex polygons in linear expected time.* Technical Report PCS-TR90-147, Department of Mathematics and Computer Science, Dartmouth College, 1990.
5. Kenneth L. Clarkson and Peter W. Shor. *Applications of random sampling in computational geometry, II.* Discrete & Computational Geometry **4**(1):387–421, December 1989.
6. Olivier Devillers, Sylvain Pion, and Monique Teillaud. *Walking in a triangulation.* Proceedings of the Seventeenth Annual Symposium on Computational Geometry (Medford, Massachusetts), pages 106–114, June 2001.

7. Herbert Edelsbrunner and Ernst Peter Mücke. *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms.* ACM Transactions on Graphics **9**(1):66–104, 1990.
8. Steven Fortune and Christopher J. Van Wyk. *Static analysis yields efficient exact integer arithmetic for computational geometry.* ACM Transactions on Graphics **15**(3):223–248, July 1996.
9. Philippe Guigue and Olivier Devillers. *Fast and robust triangle-triangle overlap test using orientation predicates.* Journal of Graphics Tools **8**(1):25–32, 2003.
10. Der-Tsai Lee and Arthur K. Lin. *Generalized Delaunay triangulations for planar graphs.* Discrete & Computational Geometry **1**:201–217, 1986.
11. Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel J. Walkington, and Han Wang. *Control volume meshes using sphere packing: Generation, refinement and coarsening.* Proceedings of the 5th International Meshing Roundtable (Pittsburgh, Pennsylvania), pages 47–61, October 1996.
12. Ernst P. Mücke. *Shapes and Implementations in Three-Dimensions Geometry.* Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993.
13. Raimund Seidel. *Voronoi diagrams in higher dimensions.* Diplomarbeit, Institut für Informationsverarbeitung, Technische Universität Graz, 1982.
14. Jonathan Richard Shewchuk. *Adaptive precision floating-point arithmetic and fast robust geometric predicates.* Discrete & Computational Geometry **18**(3):305–363, October 1997.
15. ———. *A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations.* Proceedings of the Fourteenth Annual Symposium on Computational Geometry, pages 76–85, June 1998.
16. ———. *Tetrahedral mesh generation by Delaunay refinement.* Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 86–95, June 1998.
17. ———. *Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations.* Proceedings of the Sixteenth Annual Symposium on Computational Geometry (Hong Kong), pages 350–359, June 2000.
18. ———. *Constrained Delaunay tetrahedralizations and provably good boundary recovery.* Proceedings of the 11th International Meshing Roundtable (Ithaca, New York), pages 193–204, September 2002.
19. ———. *Updating and constructing constrained Delaunay and constrained regular triangulations by flips.* Proceedings of the Nineteenth Annual Symposium on Computational Geometry, pages 181–190, June 2003.
20. ———. *General-dimensional constrained Delaunay triangulations and constrained regular triangulations I: Combinatorial properties.* Discrete & Computational Geometry **39**(1–3):580–637, March 2008.
21. Hang Si and Klaus Gärtner. *Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations.* Proceedings of the Fourteenth International Meshing Roundtable (Byron W. Hanks, editor), pages 147–163, September 2005.
22. Hang Si and Klaus Gärtner. *3D boundary recovery by constrained Delaunay tetrahedralization.* International Journal for Numerical Methods in Engineering **85**(11):1341–1364, 18 March 2011.
23. David F. Watson. *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes.* The Computer Journal **24**(2):167–172, 1981.